ТИНЬКОФФ

Стандартная библиотека JDK

Строки, работа с файлами и датами

Победённый Алексей

10.2021

Коммуникация

- Во время чтения отдельных секций не вижу реакции и комментарии.
- Будут мини перерывы на каждом слайде-отбивке и слайде с вопросами.
- Можете задавать вопросы в чате толка или поднимать руку.

Примеры слайдов с остановками

Кратко о себе

Образование

Специалист по защите информации, кандидат наук.

Карьера

Работаю Teamlead-ом в отделе Комплаенс с мая 2021.

Общий опыт разработки 20 лет.

Преподавание

Преподавал «Системное программирование на C», читал лекции на Fintech.middle и в Академии бэкенда.

Периодически выступаю на локальных митапах.

Что было на прошлой лекции?

Познакомились с основными структурами данных

Рассмотрели основные структуры данных из JCF – List, Set, Map, Queue и Stack, Tree

Stream API

Изучили Stream API, лямбды

Что ждет нас сегодня?

01 Строки

Все-все о строках. Регулярки, основные методы, UTF-8/UTF-16

02 Работа с датами и временем

Рассмотрим пакет java.util для работы с датами.

03 Вспомогательные классы работы с датами

Узнает как форматировать даты и выполнять операции с датами

04 Работа с файлами

Рассмотрим пакет java.io для работы с файлами

ТИНЬКОФФ

Строки

Методы строк (коротко)

Получение частей строки

- charAt / chars
- getBytes
- substring
- split (использует регулярные выражения)

Поиск в строке

- indexOf / lastIndexOf
- startsWith / endsWith

Создание новой строки

- valueOf
- replace / replaceAll / replaceFirst
- repeat
- strip / stripLeading / stripTrailing
- join

Сравнение

- contains
- equals / equalsIgnoreCase / contentEquals
- isBlank / isEmpty

Создание новой строки

Строки неизменяемы (immutable)

При изменении строки создается ее полная копия

Зачем

- Security. Будет не круто, если какая-то сторонняя библиотека подменит запрос к БД
- Синхронизация. Нет общего состояния – нет проблем
- Хешкод. Было бы плохо не найти в Map по ключу значение
- Перфоманс. String pool.

Создание новой строки

- valueOf – для всех примитивов
- replace / replaceAll / replaceFirst – заменяет символы
- repeat – никогда не использовал
- strip / stripLeading / stripTrailing / trim - убирает пробелы
- join – соединяет список строк, используя разделитель
- format – вставляет аргументы в строку в определенном формате. Как logger
- toLowerCase / to UpperCase – меняет регистр строки

StringBuilder/StringBuffer

StringBuilder

Класс для модификации, сборки строк. Не создает копии строки. Под капотом – char[]

```
new StringBuilder(“Test\n”)
  .append(10)
  .append(“ times”)
  .toString()
```

StringBuffer

Потокобезопасная версия StringBuilder. Коротко: используйте StringBuilder.

“123”.concat(“456”).concat(“789”)

- 123 будет скопирована 2 раза
- 456 будет скопировано 2 раз
- 789 будет скопирована 1 раз

Итого буду созданы строки:

123, 456, 789, 123456, 123456789

StringBuilder

Оптимизация

Большинство конкатенаций строк (через +) компилятор заменит на StringBuilder

С циклами проблемы

```
var s = “”;
for (int i = 0; i < 20; i++) {
  s += line + I + “\n”;
}
```

Тем не менее, в циклах ручная оптимизация

```
StringBuilder lineBuilder = new StringBuilder();
for (int i = 0; i < 20; i++) {
  lineBuilder.append("line ").append(i).append("\n");
}
```

Получение частей строки

Получение частей строки

- charAt / chars – возвращает символы строки
- substring – возвращает часть строки
- split - делит строку на части, используя регулярные выражения для задания разделителя
- lines – возвращает ленивый stream, содержащий части строки, отделенные переносом строки
- getBytes – возвращает массив байт, содержащий представление строки в указанной кодировке

Немного про кодировки

- Задают числовое соответствие символу
- Более 50 региональных кодировок, в т.ч. Windows-1251, CP866, KOI-8R для кириллицы
- Unicode – как попытка представить любой алфавит
- UTF-8 (использует 8/16/24/32 бита) и UTF-16 (использует 16/32 бита)
- Некоторые особенности UTF могут быть неожиданными

```
var wavingHandSign = "👋";
wavingHandSign.length() // 2
wavingHandSign.substring(0, 1)
```

Коротко про UTF

- **Символ – codepoint. Может занимать от 1 до 4 байт.**
- **Некоторые символы (например, флаги) кодируются с использованием нескольких codepoint и называются graphem**
- **Есть методы для работы с codepoint**

- **codePointCount** – длина строки в символах юникода
- codePointAt – возвращает символ (примерно как charAt)

Перекодирование строки

Прочитать файл в Windows-1251 или отправить как Windows-1251

```
//если друг прислал файл в кодировке Windows-1251
bytes[] contentCp1251 = file.readBytes();
String result = new String(contentCp1251, “Windows-1251”);
```

```
//если нужно выгрузить другу в кодировке 1251
bytes[] contentCp1251 = string.getBytes(“Windows-1251”);
sendFileToFriend(contentCp1251);
```

ТИНЬКОФФ

Регулярные выражения

Задачи регулярок

Валидация

Проверить, что строк соответствует сложному формату.

Частый кейс – валидация email.

Выделение части строки

Позволяет разделить строки на части, используя сложный формат.

Поиск и замена

Найти и заменить часть строки

Та самая регулярка для валидации email

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t]
)+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:
\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(
?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[
\t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\0
31]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\
](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+
(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:
(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z
|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)
?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\
r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
 \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)
?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t]
)*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
 \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*
)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t]
)+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)
*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+
|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r
\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:
\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t
]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031
]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](
?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?
:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?
:\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*)|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?
:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?
[ \t]))*"(?:(?:\r\n)?[ \t])*)*:(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\]
\000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|
\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>
```

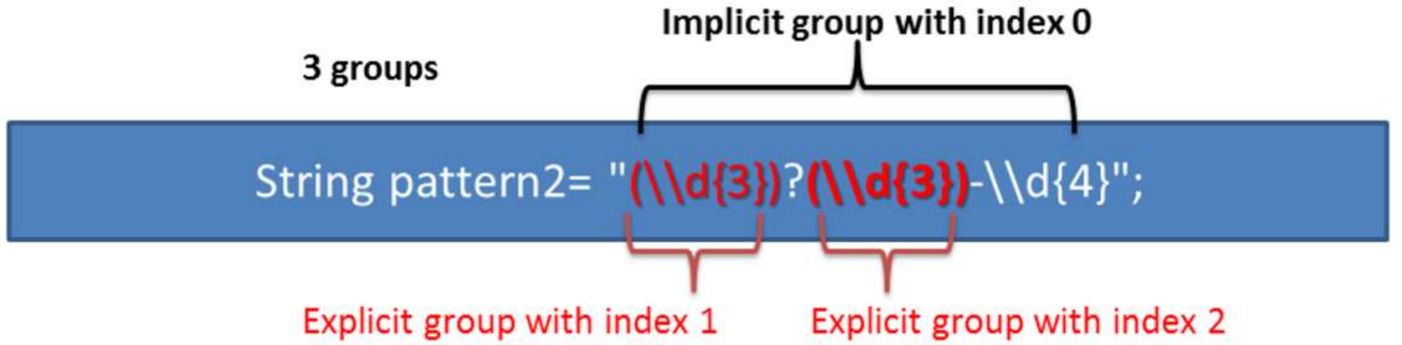
Коротко про регулярки

Описывается формат с использованием классов, модификаторов , групп и логических условий

- **\d** – однозначное число (0-9)
- **\d+** - одна или более цифр
- **\d*** - ноль или более цифр
- **\d{1, 10}** – не менее одной и не более десяти цифр
- **\w{3}** – три символа (буквы, цифры, подчеркивания)
- **\w{3,}** – не менее трех символов
- .* - любое количество любых символов
- **[a-zA-Z]** – перечисление букв с диапазоном
- **[^abc]** – все, кроме abc
- **^** - начало строки, **$** конец строки
- **0|1** – либо 0, либо 1
- **()** – группа, позволит получить содержимое того, что внутри
- **()?** – тоже, что и {0, 1}

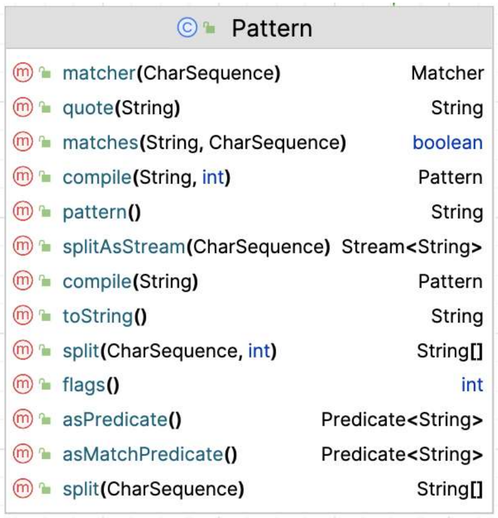
Коротко про регулярки

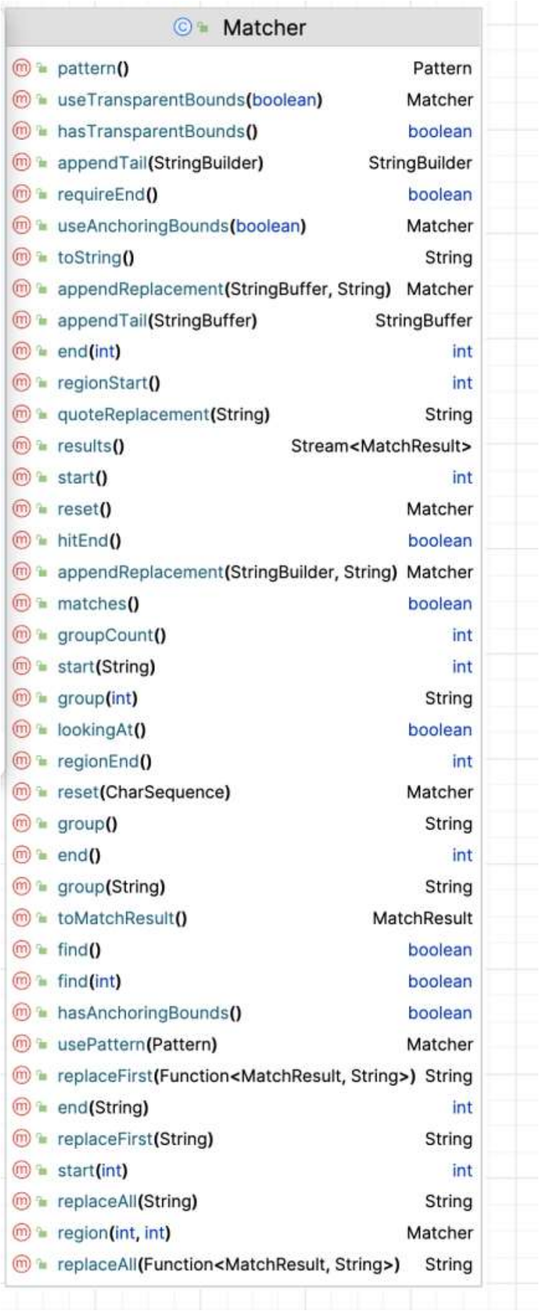
Описывается формат с использованием классов, модификаторов , групп и логических условий



123456-7898 или 456-7898 проходят валидацию
123 – значение первой группы, 456 - второй
1123456-7898 тоже пройдет валидацию, потому что нет ограничения на начало строки (достаточно добавить ^ в начало регулярки)
123456-7890**00** тоже пройдет валидацию, если не добавить $ в конец регулярки

Pattern / Matcher





Pattern / Matcher

Пример

```
Pattern pattern = Pattern.compile( regex: "(\\d-\\d{2})abcd");
Matcher matcher = pattern.matcher( input: "1-10abcd");
if (matcher.find()) {
    assert matcher.group(1) == "1-10";
}
```

Про String.split

Если просто сделать без регулярок – делайте без регулярок. Пример оптимизации в String.split (появилось в Java 7)

```
@NotNull @Contract(pure = true)
public String[] split( @NotNull String regex, int limit) {
    /* fastpath if the regex is a
     * (1) one-char String and this character is not one of the
     *     RegEx's meta characters ".$|()[{^?*+\\", or
     * (2) two-char String and the first char is the backslash and
     *     the second is not the ascii digit or ascii letter.
     */
    char ch = 0;
    if (((regex.length() == 1 &&
         ".$|()[{^?*+\\".indexOf(ch = regex.charAt(0)) == -1) ||
```

Optional

- Null-safe замена null значения
- Позволяет отличить null как результат выполнения функции, от отсутствия результата
- Как и со стримами можно собирать функциональные цепочки:

```
streamOfInts
  .findFirst()
  .map(String::valueOf)
  .orElse("0")
```

- Функция Optional#flatMap пригодится, когда значение нужно передать в другую функцию, возвращающую Optional

ТИНЬКОФФ

Вопросы?

ТИНЬКОФФ

Практика

ТИНЬКОФФ

Работа с датами. Часть 1

Почему не хранить дату строкой?

Разный формат вывод

Для разных стран/регионов будет разный формат даты

Манипуляции

Часто нужно отправить СМС на 3, 7 день или через год. Удобнее считать

Часовые пояса

Браузер присылает время по Хабаровску на сервер, который находится в московском часовом поясе и с этим нужно как-то жить

Две версии API

Date-time classes in Java	Modern class	Legacy class
Moment in UTC	java.time. **Instant**	java.util. ~~**Date**~~ java.sql. ~~**Timestamp**~~
Moment with offset-from-UTC (hours-minutes-seconds)	java.time. **OffsetDateTime**	(lacking)
Moment with time zone (`Continent/Region`)	java.time. **ZonedDateTime**	java.util. ~~**GregorianCalendar**~~
Date & Time-of-day (no offset, no zone) Not a moment	java.time. **LocalDateTime**	(lacking)

Коротко про старый API

Сейчас (UTC)

```
new java.util.Date()
new java.util.Date(System.currentTimeMillis)
```

Парсинг/форматирование

```
new SimpleDateFormat(“dd-MM-yyyy”).parse(“31-10-2023”)
new SimpleDateFormat(“dd-MM-yyyy”).format(new Date())
```

Рассчитать дату +1 день

```
Date dt = new Date();
Calendar c = Calendar.getInstance();
c.setTime(dt);
c.add(Calendar.DATE, 1);
dt = c.getTime();
```

Сравнить даты

```
var isAfter = date1.after(date2);
```

ТИНЬКОФФ

Практика

ТИНЬКОФФ

Работа с файлами

Коротко про работу с файлами

java.io.File

Представляет собой ссылку на файл или директорию

InputStream/OutputStream

Базовые интерфейсы для чтения/записи данных из файлов

FileInputStream/FileOutputStream работают непосредственно с File

new FileInputStream(new File(“test”)).readAllBytes();

new FileOutputStream(new File(“test”)).write(“Hello world”.getBytes())

InputStream/OutputStream

Обязательно закрывать после использованию

PrintWriter/BufferedReader

Предоставляют простые функции для чтения текстовых файлов

Коротко про работу с файлами

java.io.File

new File(“c:\\windows”).isDirectory(); - проверяет атрибуты объекта ФС
new File(“c:\\”).list(); - возвращает список файлов

InputStream/OutputStream

InputStream.read() возвращает следующий байт из потока. -1 означает, что все прочитано

OutputStream.write() записывает один байт (но аргумент int) или массив байт

OutputStream.flush() сбрасывает внутренний буфер. Обычно вызывается автоматом при закрытии потока.

Что будет на следующей лекции?

01 java.nio

Неблокирующий ввод вывод

02 Работа с датами и временем

Рассмотрим пакет java.time для работы с датами, временем и таймзонами

03 Работа с сетью

Блокирующая и неблокирующая работа с сетью

04 Разное

Логирование, properties-файлы, математические операции

ТИНЬКОФФ

Спасибо!